A Governance Model for Incremental, Concurrent, or Agile Projects

Dr. Alistair Cockburn
Humans and Technology

Use the model in this article to regain oversight of a complex multi-project that uses concurrent and incremental development strategies in differing amounts on different subprojects. The model makes this possible by estimating and then tracking intended versus actual functionality integrated either monthly or quarterly.

This article discusses a graphic that shows the status of larger projects, how to gather the information needed, read the status chart, and when needed, show more detailed breakdown for each subproject.

Motivation

Imagine a steering committee opening a large binder for a fairly large project, hoping to see its status. They are met by Gantt charts, cost expenditure tables, and if they are lucky, earned-value graphs. None of these quickly and accurately show the committee what is going on with the project, which components are in trouble, and where they stand with respect to delivery. Worse, many projects these days use incremental, concurrent, or agile approaches, which do not fit into the standard governance milestones of *requirements complete, design complete, code complete, and integration complete*.

This is not the article to review all the problems with that standard project governance model. The problems may, however, be briefly summarized as follows:

- The standard governance model is a single-pass waterfall model that has been roundly criticized over the years.
- It does not support incremental or concurrent approaches, both of which are being recommended more frequently these days.
- The Gantt chart does not show *percentage complete* very well, nor expected versus actual progress.
- The standard earned-value graph shows tasks complete well, but *tasks complete* does not reflect true progress on the project; true value in software development accrues suddenly (or fails suddenly) only at the moment at which the new features are integrated into the code base.

What is needed is a way that more accurately shows true value, more readily shows expected versus actual progress, and allows teams using different combinations of waterfall, incremental, concurrent, and agile approaches to show their varied strategies.

This article describes a proposed graphic that meets those constraints. Humans and Technology is starting to use it on a program that is fairly large for the private sector: about a dozen applications and two dozen supporting components and databases to be installed in various configurations in two dozen receiving sites. The graphic appears to work in our situation and has let us update a waterfall governance model to permit agile, concurrent, and incremental development.

The figures in this article are simplifications of the more complicated versions created for that program, so we have a sense that they scale reasonably. We see ways to update the graphic automatically, or even to update manually monthly for the steering committee meeting. While not fully tested, it shows promise.

Here are the details – please write me if you try it out and come up with improvements. (Note: The figures in this article are printed in black and white; the online version [1] uses color.)

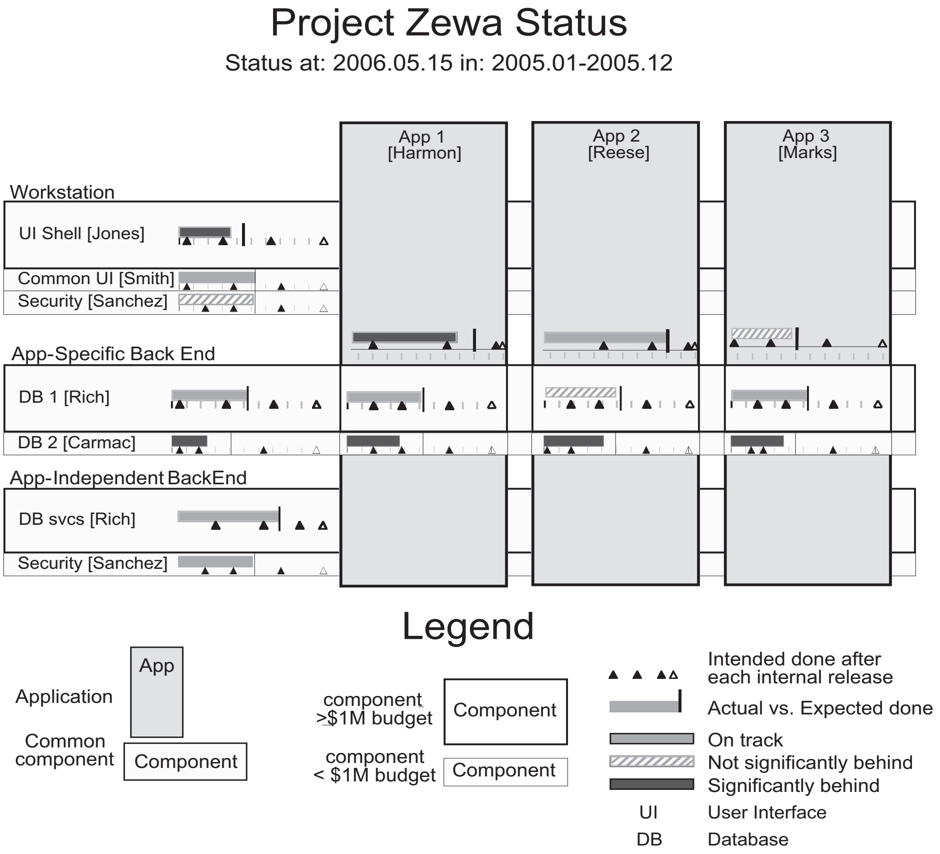
The Graphic

The graphic contains the following elements:

- The components being built, how each fits into the system architecture, and the person responsible for each.
- Codependency relations between horizontal and vertical components.
- The incremental growth strategy for each component over the project life.
- The expected versus actual amount of work completed to date.
- A color or shading to mark the alarm level for each component.

The following is a discussion of those topics in three categories: the system structure, the intended strategies, and

Figure 1: *Governance Graphic*



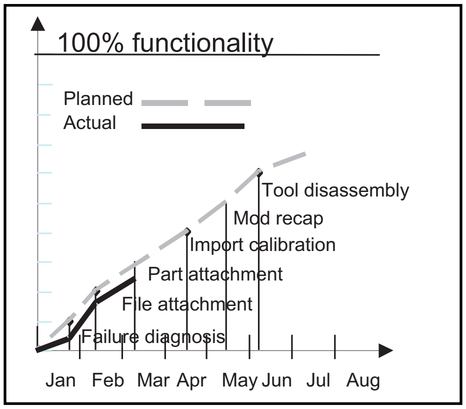


Figure 2: *Burn-Up Chart*

expected versus ideal progress.

System Structure

In Figure 1, the three vertical rectangles indicate *applications* – the items bought separately for end-user functionality. The seven horizontal rectangles indicate service components needed across applications – on the user's desktop or the back end. Two of the horizontal components cross *in front of* the applications to indicate that the horizontal component contains a specific, different functionality or data for each application. Domain databases that get extended for each new application are likely to be among these application-specific, back-end components (more on this later).

The system shown in Figure 1 is fairly simple. The first project for which I drew this graphic had 17 horizontal and 15 vertical components, and additional coloring to show legacy components that were to be removed over time. We were still able to draw it legibly on legal-sized paper.

Intended Strategies

Although incremental development has been around much longer than the agile movement, the question of how to show different incremental strategies for governance purposes – preferably in a small space – is still open.

Figure 3: *Target Progress Markers*

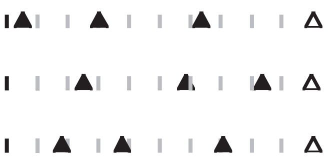
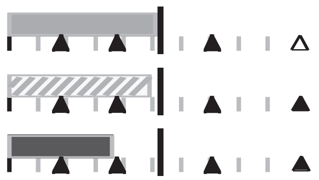


Figure 4: *Intention-Completion Bars*



Agile project teams measure progress not according to how many requirements have been gathered, but by how much running functionality has been designed, programmed, and integrated (Ron Jeffries neatly calls these running tested features, or RTF [2]). A common way to show the growth of RTF is through burn-up charts, as in Figure 2, which shows the expected versus actual integration of a set of workflow components by month.

Agile burn-up charts are very similar to traditional earned-value charts with one crucial exception: The team only gets credit when the features are integrated into the full code base and the result passes testing [3]. This single chart shift makes a big difference in the reliability of the progress data presented.

Burn charts show more than we need and take too much space for governance oversight purposes. To reduce their size and information, we use the idea of an *internal release* (IR).

A team that cannot deploy its system to live users every few months can *pretend* to deploy the system. It can test, integrate, and deploy the system to the computer of one or two *friendly* but real users. The team thus exercises the end of their development process and gets true user feedback. Putting the system in front of real users (as opposed to a test machine in the basement) motivates the development team to take their work seriously.

Such an IR should happen every one, two, or three months. There are many reasons not to deploy fully every three months, but there is almost no reason not to carry out an IR. These IRs fit neatly into a monthly or quarterly reporting mechanism.

With RTFs and IRs, we are ready to capture various development strategy or strategies that might show up on an incremental development project.

In Figure 3, the vertical tick-marks show 10 percent units of completed RTF from left to right (100 percent complete at the right). The triangle milestone markers show the amount of RTF the team targets to have completed and integrated at each IR milestone. Figure 3 shows three teams' strategies as follows:

- The top team plans to get less than 10 percent of its functionality in place in the first IR, and to add functionality in roughly equal thirds after that.
- The middle team intends to get 25 percent done in the first quarter, 60 percent by the end of the second quarter, and almost 85 percent through the third quarter (possibly so they have time to fix mistakes in the fourth quarter).
- The bottom team expects to get almost 20 percent completed and integrated in each of the first two quarters, and then to speed up and get 30 percent done in each of the last two quarters.

Alert readers will notice that these tickmark drawings capture the vertical axis of the burn-up charts at the IR times.

These small diagrams let different teams work in different ways and report on their intentions. This is our goal.

Expected Versus Ideal Progress

Intention-Completion Bars

Figure 4 adds to Figure 3 the work actually completed compared to the work targeted for any point in time.

In Figure 4, the taller vertical bar moves from left to right within each IR period to show the current targeted accomplishment. It can run at a constant rate within each period according to the calendar, or it can be synchronized with the team's iteration plans (two- to six-week planning and tracking time windows). The shaded rectangles show the functionality (RTF) completed and integrated to date.

In Figure 4, we see that the top team is delivering according to schedule, the middle team is a little behind, and the bottom team still has not finished the work scheduled for the second IR.

Two comments must be made at this point about RTF. The first is that not all final deliverables consist of *features* that *run*. Content databases and end-user documentation are examples. Teams can create intention-completion bars for whatever their final deliverables are, since those bars show growth of accomplishment over time.

The second comment is that measures not tied to RTF are naturally hazardous since it is so easy to start tracking completion of artifacts that do not directly get bought by customers. Linking accomplishments to RTF makes the reporting of actual *value* both easier and more accurate.

Application-Specific Components

Horizontal components such as application databases require new work and new content for each new application. Progress on these application-specific horizontal components is typically difficult to report on since *where they are* varies from application to application.

To show the status of such a component, we use intention-completion bars for the independent portion of the component and for each application it must serve. This lets the teams move at different speeds as suits their particular situations, and allows the steering committee to see each team's status.

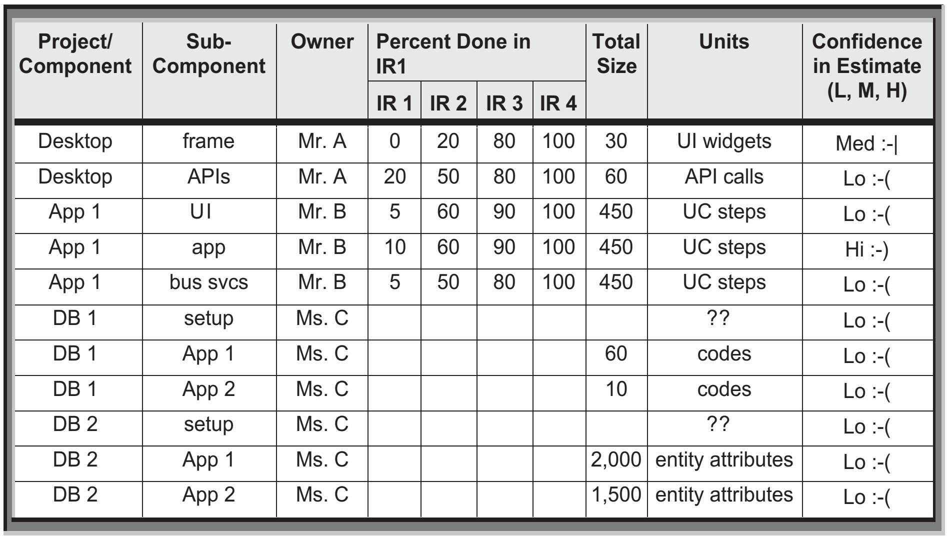
Summarizing the Graphic

Let us review the elements of the graphic briefly:

- The rectangles represent components, subsystems, or applications. Vertical rectangles show applications; horizontal ones show components that get used across multiple applications (this could be reversed for better layout if, for example, there are many applications and only a few cross application components).
- Each rectangle shows the place of the component in the overall architecture: The top set of horizontal components reside on the desktop, the middle and bottom sets of horizontal components reside as back-end services. The horizontal rectangles running *behind* the applications get created independently of the applications; the horizontal rectangles running *in front of* the applications require application-specific work or content.
- Intention-completion markers are created for each component. They show the percentage of RTF intended for completion at each IR milestone, the expected and the actual current accomplishment, and the alarm level. Intention-completion bars are created for each component and for each intersection of application-dependent components.

Collecting the Information

The information rendered in Figure 1 also fits into a spreadsheet, a more useful place to keep it while gathering and updating the information. We can use automated tools to gather information about each component every week or two, and roll up each team's accomplishments into reports at various levels. The highest level is the one that gets *painted* onto the graphic either by hand or automatically. (The graphic can be generated automatically using graphic markup languages, but that programming effort may take longer than simply coloring the bars each month).



Project/ Component	Sub-Component	Owner	Percent Done in IR1				Total Size	Units	Confidence in Estimate (L, M, H)
			IR 1	IR 2	IR 3	IR 4			
Desktop	frame	Mr. A	0	20	80	100	30	UI widgets	Med :-|
Desktop	APIs	Mr. A	20	50	80	100	60	API calls	Lo :-(
App 1	UI	Mr. B	5	60	90	100	450	UC steps	Lo :-(
App 1	app	Mr. B	10	60	90	100	450	UC steps	Hi :-)
App 1	bus svcs	Mr. B	5	50	80	100	450	UC steps	Lo :-(
DB 1	setup	Ms. C						??	Lo :-(
DB 1	App 1	Ms. C					60	codes	Lo :-(
DB 1	App 2	Ms. C					10	codes	Lo :-(
DB 2	setup	Ms. C						??	Lo :-(
DB 2	App 1	Ms. C					2,000	entity attributes	Lo :-(
DB 2	App 2	Ms. C					1,500	entity attributes	Lo :-(

API - Application Program Interface, App - Application, UC - User Class

Table 1: *Estimating Spreadsheet*

Gathering the Estimates

It is one thing to say, "We intend to be 20 percent done after the first internal release," but the steering committee needs to know, "Twenty percent of what?" Being behind on 20 percent of two use cases is very different than being behind on 20 percent of 80 use cases.

> "A team that cannot deploy its system to live users every few months can pretend to deploy the system. It can test, integrate, and deploy the system to the computer of one or two friendly but real users."

To capture the *of what* for tracking, we need three pieces of information. The first, "What is the unit of accomplishment?" often consists of use cases, or more likely, individual steps in use cases. Sometimes something quite different is appropriate. A desktop component might have as units of accomplishment user interface (UI) widgets (frames, pull-down lists, buttons) and interface calls used by the applications. A database might have entities and attributes, a Web site might have articles and images, a medical database might have medical codes as a unit of accomplishment.

The second piece of information is, obviously, "About how many units do you expect to create?"

The third piece of information is the confidence level on the estimate. At the beginning of the project, it is appropriate to have low confidence ratings in the estimates: "We expect somewhere between 15 and 50 UI widgets, call it 30, plus or minus 50 percent;" however, that comes with the caution, "You called me into this room and made me give you numbers, but it's not like I have a really good basis for those numbers!"

The initial rough-size estimate is still useful for getting an early handle on the size and shape of the thing to be built. That is why the information is collected even when the confidence rating is low. Marking a low confidence rating is useful to the project leaders because they can then raise the priority of getting enough information to improve the confidence level.

Needless to say, the estimate should be updated at the start of successive iterations with raised expectations about its accuracy and confidence levels.

Table 1 shows a spreadsheet that can be used to capture the estimates. Note that the confidence rating is accompanied by a smiling, neutral, or frowning face to visually tag this important information.

Gathering the Status

To tag the timeline, we need to give each iteration or planning window a milestone number such as an IR completed then followed by iteration completed. Thus, milestone 0.3 means the end of the third iteration before the first IR, and milestone 2.1 means the end of the first iteration after the second IR.

After iterations, the teams send in

Project/ Component	Sub- Component	Owner	Percent Done In				Total Size	Units	Expected at IR 2.3 (percent)	Expected at IR 2.3 (units)	Actual at IR 2.3 (units)
			IR 1	IR 2	IR 3	IR 4					
Desktop	frame	Mr. A	0	20	80	100	30	UI widgets	50%	15	15
Desktop	APIs	Mr. A	20	50	80	100	60	API calls	65%	39	36
App 1	IU	Mr. B	5	60	90	100	450	UC steps	75%	337	310
App 1	app	Mr. B	10	60	90	100	450	UC steps	75%	337	320
App 1	bus svcs	Mr. B	5	50	80	100	450	UC steps	65%	292	280

Table 2: *Summary Spreadsheet*

their RTF numbers, which get rolled up into a summary spreadsheet at any level of granularity desired. The nice thing here is that this roll-up can be produced automatically with simple tools. Table 2 shows how the first few rows of such a spreadsheet might look after iteration 2.3.

The Status Report Packet

The graphic in Figure 1 serves as a good *summary page* of the package put in front of the steering committee. That package also needs detail pages for the separate subprojects.

Table 3 shows a sample detail page. This detail page has three sections after the header:

- A status/targeted/deferred and risk snapshot for each section of work within the component.
- A commentary, including surprises and lessons learned during the previous period.
- Cost roll-up information.

The most unusual part of this status page is the way in which the intention-completion bars are constructed to describe the strategy and accomplishments of non-RTF work.

Intention-Completion Bars for Non-RTF Work

When someone sees a component marked with a high-alarm status bar on the summary page, they will naturally want to read more detail. They will need to understand what is happening with respect to requirements gathering, UI design, design and programming, and user documentation.

Figure 5: *A Sequential Development Strategy*

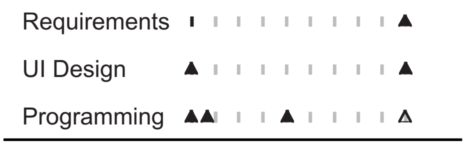
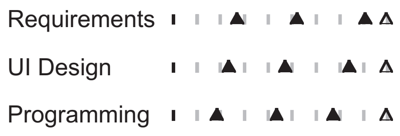


Figure 6: *A Concurrent Development Strategy*



The good news is that we can use the intention-completion bars to show progress within each specialty, whether the team is using a sequential (waterfall) strategy or a concurrent strategy. Figures 5 and 6 illustrate the two.

Figure 5 shows a team planning to work in sequential fashion. They plan to finish all their requirements in the first period. They do not plan on starting either the UI design or the programming in that period. They expect to get the UI design fully complete in the second quarter. They plan to get perhaps 10 percent of the programming done in the second quarter, and the rest done equally in the third and fourth quarters.

Figure 6 shows a strong concurrent strategy. This team plans to get not quite a third of their requirements settled in the first period, and to have nearly as much UI design and programming done as requirements gathered. The requirements people will lead the UI design people by a small amount, and the UI design people will lead the programmers by a small amount, but otherwise these groups will run parallel to each other. They intend to continue in this fashion throughout the entire project.

In this article, I do not wish to indicate that either approach is superior to the other. What is important here is that both sequential and concurrent strategies (and many combinations) can be shown using the intention-completion bars.

Status, Targeted, Deferred, and Risks

For any component, the steering committee members will want to see the following at the top of the detail page:

- The intention-completion bars for the whole component from the summary sheet, and for the work efforts within the component, including non-RTF work as just described.
- What was targeted for accomplishment during this reporting period?
- What work is being deferred from this period into the next?
- The dominant problems each subteam is facing or the risks they expect.

The risks and problems column lets the team signal for help, whether that means more people, more equipment, more time with customers, etc.

Surprises, Lessons Learned, Items Needing Special Attention

The middle of the page allows the team to reflect and report on what happened during the reporting period.

The first section describes the surprises discovered. On projects I have visited, these have included the programmers not getting as much done as expected, a piece of technology not working as expected, or, conversely, a new practice such as daily stand-up meetings being effective.

The second section describes the lessons to be taken out of the period's work. These might include multiplying developer estimates by a factor before committing to them, doing technology spikes before committing to technology, or choosing to keep the new, daily, stand-up meetings. These must be truly lessons *learned* within the period, not speculations on what might work in the future.

The third section is for anything the team wishes to report on. It may expand on risks or highlight some particular worry to which they will be paying close attention.

Cost Roll-up

Finally, the steering group needs to see how fast the money is being used. This section may be presented in tabular or burn-up form, and include staffing sizes as well as budget information as desired.

Summary

The first contribution of this article is the description of the intention-completion graphic, showing the following:

- The strategy that the team has in mind for its work, whether sequential or concurrent.
- How much the team had expected to have done at this reporting point.
- How much the team actually has done at this point.

The intention-completion graphic is important because it allows different teams to choose different strategies and report on them, all in the same format. The absence of a common reporting format has been a painful point for incremental, concurrent, and agile projects for a long time.

The second contribution is the project summary graphic and its spreadsheet counterpart. The spreadsheet allows the leadership team to collect estimates and plans at a very early point in the project, and easily update these by using automated tools. The graphic provides a way to show at a glance the entirety of a quite complex project. This addresses the questions, "What are we building?" and "How are we doing?"

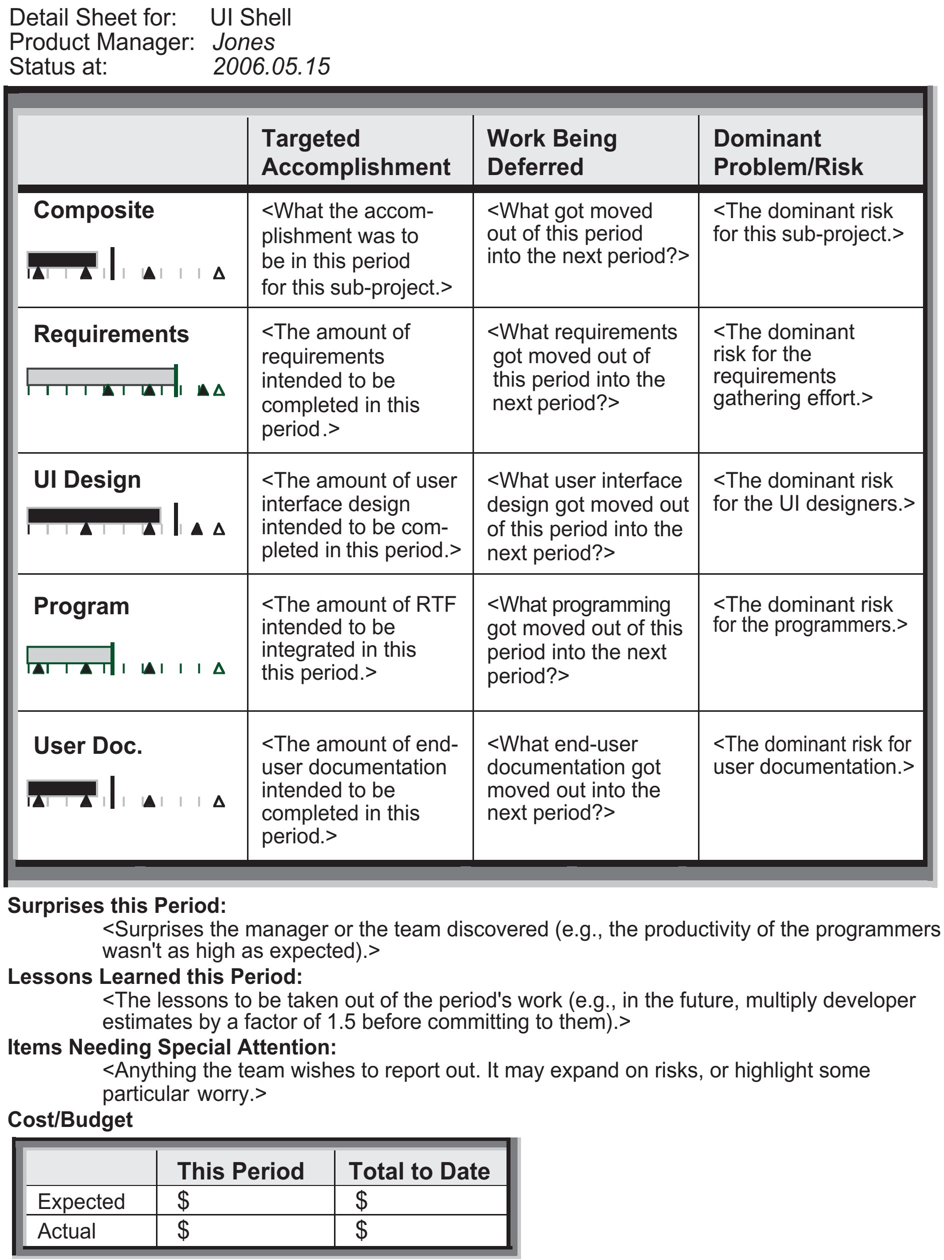
The third contribution is the description of a sample, one-page detail sheet (see Table 3) for each component or subproject. This page shows at a glance the strategies and status within the subproject, along with key information the steering committee needs to understand and respond to.

The resulting packet of information allows people who meet only once a month or quarter to assess the intentions and status of projects that use various mixtures of waterfall, incremental, concurrent, and agile strategies.

If you use this model and find ways of improving it, please let me know at <acockburn@aol.com.>◆

References

1. Cockburn, A. "A Governance Model for Incremental, Concurrent, or Agile Projects." CrossTalk Feb. 2006 <www.stsc.hill.af.mil/crosstalk/2006/02/0602Cockburn.html>.
2. Jeffries, R. "A Metric Leading to Agility." XProgramming.com 14 June 2004 <www.xprogramming.com/xpmag/jatRtsMetric.htm>.
3. Cockburn, A. "Earned Value and Burn Charts." Technical Report. Humans and Technology, Apr. 2004 <http://alistair.cockburn.us/crystal/articles/evabc/earnedvalueandburncharts. htm>.



Detail Sheet for: UI Shell
Product Manager: *Jones*
Status at: *2006.05.15*

	Targeted Accomplishment	Work Being Deferred	Dominant Problem/Risk
Composite	<What the accomplishment was to be in this period for this sub-project.>	<What got moved out of this period into the next period?>	<The dominant risk for this sub-project.>
Requirements	<The amount of requirements intended to be completed in this period.>	<What requirements got moved out of this period into the next period?>	<The dominant risk for the requirements gathering effort.>
UI Design	<The amount of user interface design intended to be completed in this period.>	<What user interface design got moved out of this period into the next period?>	<The dominant risk for the UI designers.>
Program	<The amount of RTF intended to be integrated in this this period.>	<What programming got moved out of this period into the next period?>	<The dominant risk for the programmers.>
User Doc.	<The amount of end-user documentation intended to be completed in this period.>	<What end-user documentation got moved out into the next period?>	<The dominant risk for user documentation.>

Surprises this Period:
<Surprises the manager or the team discovered (e.g., the productivity of the programmers wasn't as high as expected).>
Lessons Learned this Period:
<The lessons to be taken out of the period's work (e.g., in the future, multiply developer estimates by a factor of 1.5 before committing to them).>
Items Needing Special Attention:
<Anything the team wishes to report out. It may expand on risks, or highlight some particular worry.>
Cost/Budget

	This Period	Total to Date
Expected	$	$
Actual	$	$

Table 3: *Detail Sheet for UI Shell*

About the Author

Alistair Cockburn, Ph.D., is an internationally respected expert on object-oriented design, software development methodologies, use cases, and project management. The author of two Jolt Productivity award-winning books, "Agile Software Development" and "Writing Effective Use Cases," as well as the perennial favorite, "Surviving OO Projects," he was one of the authors of the Agile Development Manifesto. Cockburn defined an early agile methodology for the IBM Consulting Group in 1992, served as special advisor to the Central Bank of Norway in 1998, and has worked in companies from Scandinavia to South Africa, North America to China. Internationally, he is known for his seminal work on methodologies and use cases, as well as his lively presentations and interactive workshops. Many of his materials are available online at <http://alistair.cockburn.us>.

Humans and Technology
1814 Fort Douglas CIR
Salt Lake City, UT 84103
Phone: (801) 582-3162
E-mail: acockburn@aol.com